Lab 5: MNIST/FMNIST

2024-02-25

Homework (lab 4 or lab 5)

> The homework consists of questions to be solved using R or python commands. You can choose between lab 4 and lab 5. You will type the necessary commands to solve the questions and the necessary comments to understand your calculations in a `pdf/html` file that you will save under the name TML_yourname. Your homework is to be sent via EPI before Monday February 26, 5pm (before class).

The MNIST database is a well-known database of handwritten digit images (0-9) that is used to evaluate machine learning algorithms. The database contains 60,000 training images and 10,000 test images. The MNIST data is stored in 4 binary files, which can be difficult to manage directly. Download the data (directly on Yann LeCun's website here or here for Fashion-MNIST) and you can use the code below to load them under `R`. In python, they (MNIST and FMNIST) are accessible via the tensorflow API for example.

```
rm(list = ls())

library(foreach)
library(doParallel)
```

```
## Ładowanie wymaganego pakietu: iterators
```

```
## Ładowanie wymaganego pakietu: parallel
```

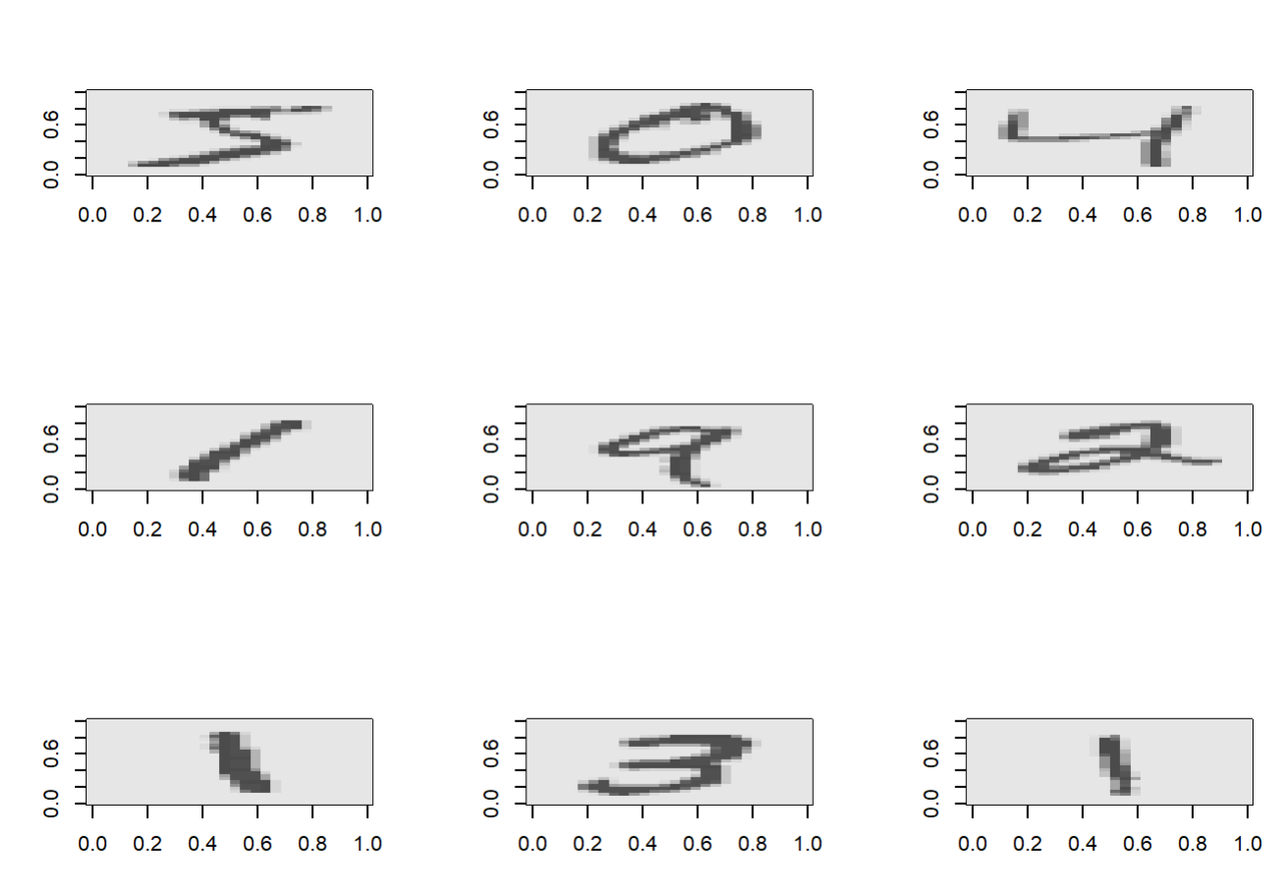
```
library(stringr)
library(e1071)
library(ggplot2)

setwd("C:/Machine_Learning_A/Final_Project")

nTrain <- readBin(con = "train-images.idx3-ubyte", what = "integer", n = 4,
                  size = 4, endian = "big")[2]
nTest <- readBin(con = "t10k-images.idx3-ubyte", what = "integer", n = 4,
                 size = 4, endian = "big")[2]
nRows <- readBin(con = "train-images.idx3-ubyte", what = "integer", n = 4,
                 size = 4, endian = "big")[3]
nCols <- readBin(con = "train-images.idx3-ubyte", what = "integer", n = 4,
                 size = 4, endian = "big")[4]
# Load training data
images.train <- array(as.numeric(readBin(con = "train-images.idx3-ubyte",
                                         what = "raw",
                                         n = nTrain * nRows * nCols + 16,
                                         endian = "big")[-(1:16)]),
                      dim = c(nCols, nRows, nTrain))
labels.train <- as.numeric(readBin(con = "train-labels.idx1-ubyte",
                                   what = "raw",
                                   n = nTrain + 8,
                                   endian = "big")[-(1:8)])
# Load test data
images.test <- array(as.numeric(readBin(con = "t10k-images.idx3-ubyte",
                                        what = "raw",
                                        n = nTest * nRows * nCols + 16,
                                        endian = "big")[-(1:16)]),
                     dim = c(nCols, nRows, nTest))
labels.test <- as.numeric(readBin(con = "t10k-labels.idx1-ubyte",
                                  what = "raw",
                                  n = nTest + 8,
                                  endian = "big")[-(1:8)])
```

0. Definitions of function used later on:

```
set.seed(12)
#Functions dataPrep and createMatrix are used to manipulate data (sample etc.)
dataPrep <- function(p, n, Data_Images, Data_Labels){ #p is portion of Data to keep, n is the pa
rameter which indicates which
                                                       #labels to keep

  sample_size <- length(Data_Images[1,1,]) * p

  small_sample <- sample(length(Data_Images[1, 1, ]),
                         size = sample_size)

  Images <- Data_Images[,,small_sample]

  Labels <- Data_Labels[small_sample]

  subs_train <- which(Labels <= n) #Choosing data which has label 0 1 or 2

  Images <- Images[,,subs_train]

  Labels <- Labels[subs_train]

  ret_list <- list(Images, Labels)

  return(ret_list) #Return is an array with first element with Images and the second with Labels
}

createMatrix <- function(Data_List){

  MatrixTrain <- matrix(0, nrow = length(Data_List[[1]][1,1,]),
                        ncol = length(c(Data_List[[1]][,,1])))

  for(j in 1:length(Data_List[[1]][1, 1, ])){
    MatrixTrain[j,] <- c(Data_List[[1]][, , j])
  }

  Matrix_Train <- MatrixTrain/255

  Labels_Train <- as.factor(Data_List[[2]])

  ret_list <- list(Matrix_Train, Labels_Train)

  return(ret_list)
}

#Functions fitAndTest and benchmarkValues are used to compare different parameters of a models.

fitAndTest <- function(train_data, test_data, cost_in = 1, gamma_in = 0.05){

  svm_fit <- e1071::svm(train_data[[1]],
                        train_data[[2]],
                        kernel = "radial",
                        cost = cost_in,
                        gamma = gamma_in,
                        scale = TRUE)

  ypred <- predict(svm_fit, test_data[[1]])

  conf_mat <- table(predict = ypred, truth = test_data[[2]])

  corr_pred <- 0

  incorr_pred <- 0

  for(i in 1:ncol(conf_mat)){
    for(j in 1:ncol(conf_mat)){
      if(j == i){
        corr_pred <- corr_pred + conf_mat[i,j]
      }
      else
      {
        incorr_pred <- incorr_pred + conf_mat[i,j]
      }
    }

  }

  Total_MissClass_Rate <- corr_pred/(corr_pred + incorr_pred)

  return(Total_MissClass_Rate)
}

benchmarkValues <- function(Data_Imag_Train, Data_Lab_Train, Data_Imag_Test, Data_Lab_Test, n){
  #n is the number of labels we are considering
  #(e.g. 0, 1, 2 -> n=2)

  MissClassVector <- vector(mode = "double", length = n)

  sample_Train_inc <- ((1/2)-(1/30))/8

  sample_Test_inc <- ((1/2)-(1/10))/8

  for(i in 2:n){

    p_train <- 1/30 + sample_Train_inc * (i-2)

    Train_Prep <- dataPrep(p_train, i, Data_Imag_Train, Data_Lab_Train)

    Train_Data <- createMatrix(Train_Prep)

    p_test <- 1/10 + sample_Test_inc * (i-2)

    Test_Prep <- dataPrep(p_test, i, Data_Imag_Test, Data_Lab_Test)

    Test_Data <- createMatrix(Test_Prep)

    MissClassVector[i] <- fitAndTest(Train_Data, Test_Data)

  }

  return(MissClassVector)
}
```

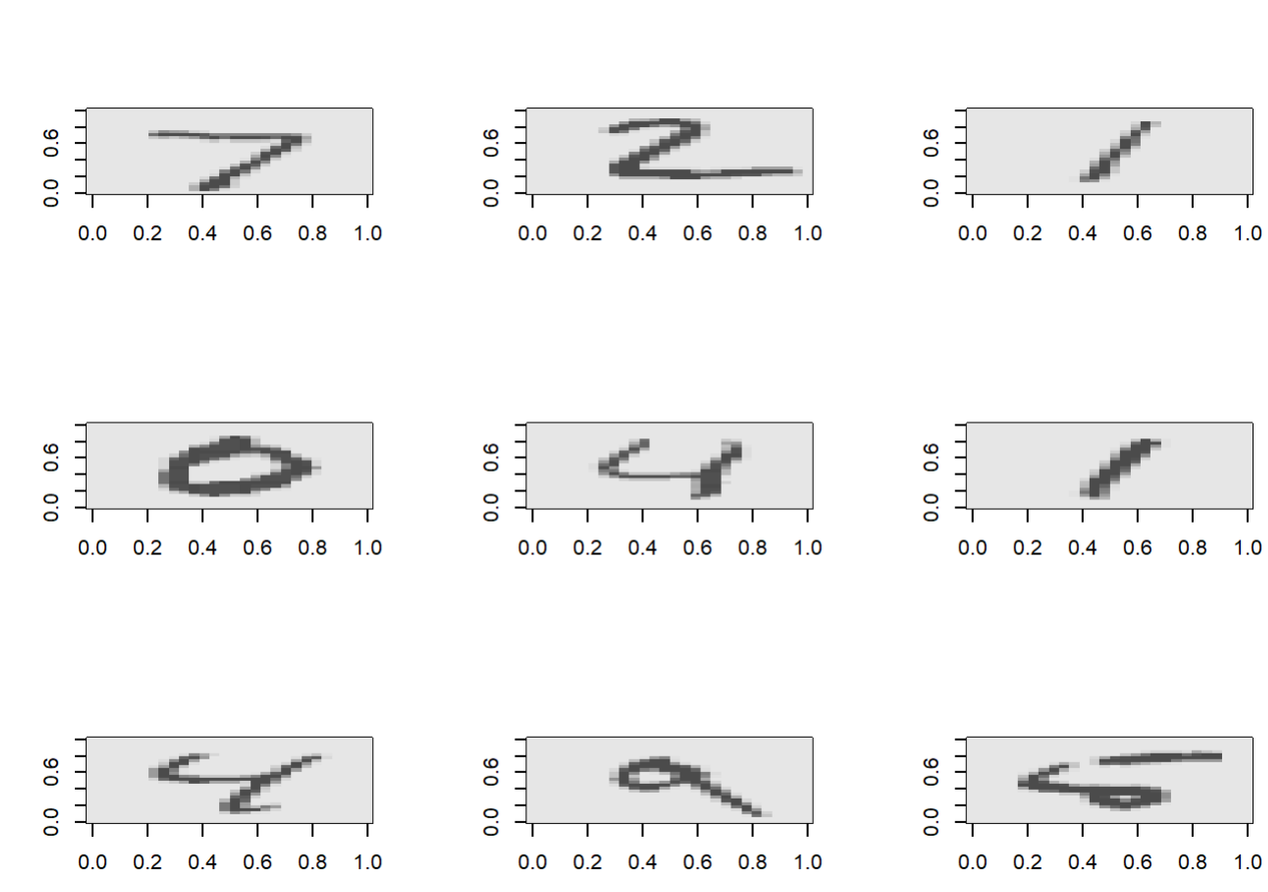
1. Display the first 10 images of the training base. You can use the functions `image` and `grey` or the pacakge `imager` (by converting them as `cimg` objects). With Python `matplotlib`.

```
par(mfrow = c(3, 3))

v <- c(1:28) # We use v to inverse the order of the rows (so that images are familiar to humans)

for(i in 1:9){
  image(images.train[, , i][, rev(v)], col = grey.colors(n = 255, rev = TRUE))
}
```



2. Same question for the first 10 images of the test base

```
par(mfrow = c(3, 3))
for(i in 1:9){

  image(images.test[,,i][, rev(v)], col = grey.colors(n = 255, rev = TRUE))

}
```



3. First, reduce the size of the database (for reasons of computation time). Create a random subset of the training base of 30 times smaller size. We also restrict ourselves to 3 classes (0,1,2).

```
Train_Prep <- dataPrep(1/30, 2, images.train, labels.train)
```

4. Prepare the data for SVM (normalize them). For example, you can put the training data in a matrix (of suitable size).

```
Train_Data <- createMatrix(Train_Prep)
```

5. Train an SVM on the training data (e.g. you can use the `svm` function from the `e1071` package under `R`, `sklearn.svm.SVC` under python). Again, for CPU time reasons, we set the parameters a priori at first. Use a gaussian kernel $K(u, v)$

```
svm_fit <- e1071::svm(Train_Data[[1]],
                      Train_Data[[2]],
                      kernel = "radial",
                      cost = 1,
                      gamma = 0.05,
                      scale = TRUE)
```

6. Reduce the size of the test data (e.g., keep only the first 1000 images) and prepare them in the same way as the training data.

```
Test_Prep <- dataPrep(0.1, 2, images.test, labels.test)

Test_Data <- createMatrix(Test_Prep)
```

7. Calculate the prediction error of the svm (on the first three classes only).

```
ypred <- predict(svm_fit, Test_Data[[1]])

conf_mat <- table(predict = ypred, truth = Test_Data[[2]])

corr_pred <- 0

incorr_pred <- 0

for(i in 1:ncol(conf_mat)){
  for(j in 1:ncol(conf_mat)){
    if(j == i){
      corr_pred <- corr_pred + conf_mat[i,j]
    }
    else
    {
      incorr_pred <- incorr_pred + conf_mat[i,j]
    }
  }

}


Total_MissClass_Rate <- corr_pred/(corr_pred + incorr_pred)

cat("The accuracy of our model is:", Total_MissClass_Rate)
```

```
## The accuracy of our model is: 0.9840764
```

8. Display the confusion matrix.

```
conf_mat
```

```
##        truth
## predict   0   1   2
##       0 100   0   3
##       1   0 116   1
##       2   0   1  93
```

9. Compare the results using the classification method of your choice (e.g. CNN).

```
library(keras)
# Load the MNIST dataset
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

# Preprocess the data
x_train <- array_reshape(x_train, c(nrow(x_train), 28, 28, 1))
x_test <- array_reshape(x_test, c(nrow(x_test), 28, 28, 1))
x_train <- x_train / 255
x_test <- x_test / 255
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)

# Define the CNN model architecture
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = 'relu',
                input_shape = c(28, 28, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 10, activation = 'softmax')

# Compile the model
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

# Train the model
history <- model %>% fit(
  x_train, y_train,
  epochs = 5,
  batch_size = 128,
  validation_split = 0.2
)
```

```
## Epoch 1/5
## 375/375 - 15s - loss: 0.3148 - accuracy: 0.9032 - val_loss: 0.0771 - val_accuracy: 0.9786 - 1
5s/epoch - 40ms/step
## Epoch 2/5
## 375/375 - 15s - loss: 0.0972 - accuracy: 0.9718 - val_loss: 0.0504 - val_accuracy: 0.9852 - 1
5s/epoch - 39ms/step
## Epoch 3/5
## 375/375 - 14s - loss: 0.0678 - accuracy: 0.9803 - val_loss: 0.0418 - val_accuracy: 0.9880 - 1
4s/epoch - 37ms/step
## Epoch 4/5
## 375/375 - 15s - loss: 0.0551 - accuracy: 0.9831 - val_loss: 0.0403 - val_accuracy: 0.9883 - 1
5s/epoch - 39ms/step
## Epoch 5/5
## 375/375 - 14s - loss: 0.0461 - accuracy: 0.9861 - val_loss: 0.0378 - val_accuracy: 0.9891 - 1
4s/epoch - 37ms/step
```

```
# Evaluate the model
score <- model %>% evaluate(x_test, y_test, verbose = 0)
cat('Test loss:', score[[1]], '\n')
```

```
## Test loss: 0.02786614
```

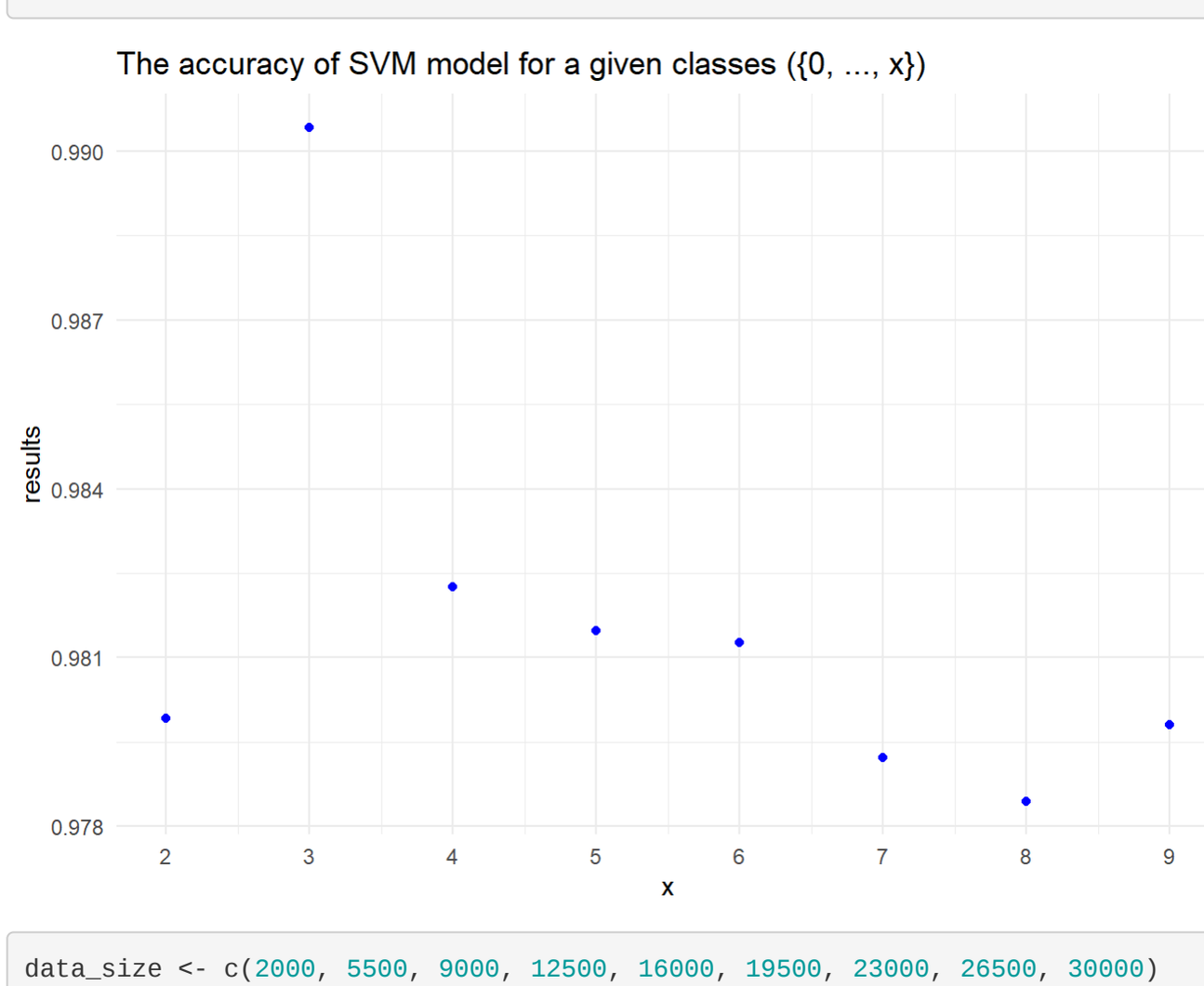
```
cat('Test accuracy:', score[[2]], '\n')
```

```
## Test accuracy: 0.9915
```

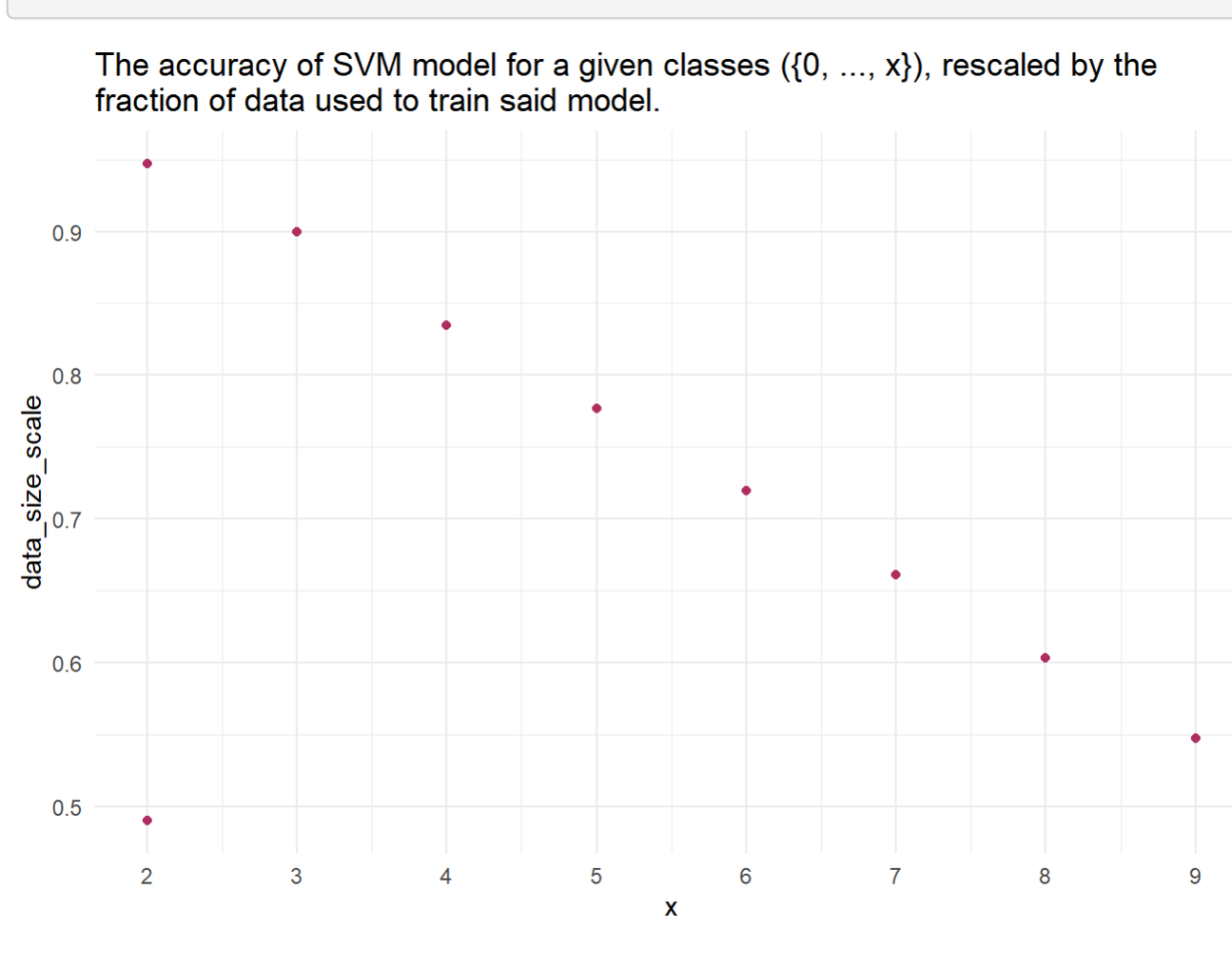
As we can see, the Accuracy for a Convolutional Neural Network is nearly 99% for 10 classes.

10. Test the performance of the SVM by considering other classes (more difficult to distinguish a priori and by increasing their numbers). You can also compare your results with another classifier.

```
#results <- benchmarkValues(images.train, labels.train, images.test, labels.test, 10)

results <- c(0.9901961, 0.9799073, 0.9904306, 0.9822485, 0.9814815, 0.9812613, 0.9792244, 0.9784
444, 0.9798000) # I stored the original results to avoid computing them again

results <- results[2:length(results)]

x <- c(2:9)

Graph_Data <- as.data.frame(cbind(x,results))

ggplot(Graph_Data, aes(x, results)) +
  geom_point(col = "blue") +
  scale_x_continuous(breaks = 2:9) +
  labs(title = "The accuracy of SVM model for a given classes ({0, ..., x})") +
  theme_minimal()
```




```
data_size <- c(2000, 5500, 9000, 12500, 16000, 19500, 23000, 26500, 30000)

data_size <- 1 - (data_size * 1/60000)

data_size_scale <- data_size * results

Graph_Data_rescale <- as.data.frame(cbind(x, data_size_scale))

long_title <- "The accuracy of SVM model for a given classes ({0, ..., x}), rescaled by the frac
tion of data used to train said model."

wrapped_title <- str_wrap(long_title, width = 80)  # Adjust the width as needed

ggplot(Graph_Data_rescale, aes(x, data_size_scale)) +
  geom_point(col = "maroon") +
  scale_x_continuous(breaks = 2:9) +
  labs(title = wrapped_title) +
  theme_minimal()
```

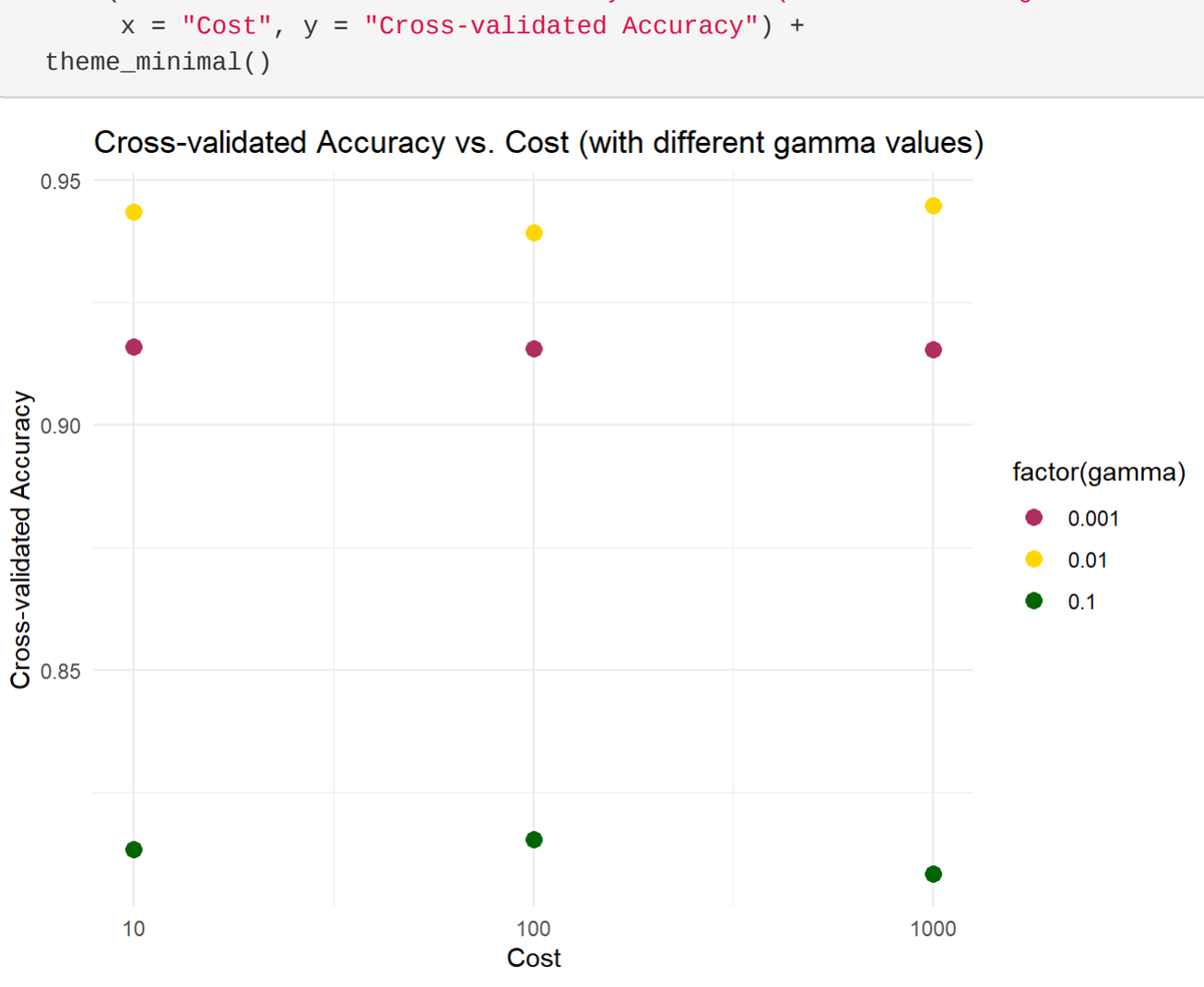


11. Using the pacakges foreach and doMC (you can consult the related vignettes foreach or doMC) parallelize the calibration step by K-fold cross-validation of the parameters γ and `cost` of the SVM. We will take for example $\gamma = cost = 10^{-3:3}$ and $K = 5$. In python, the [multiprocessing] module (https://docs.python.org/2/library/multiprocessing.html), and in particular the `Pool` class (with its associated methods, .apply, .map).

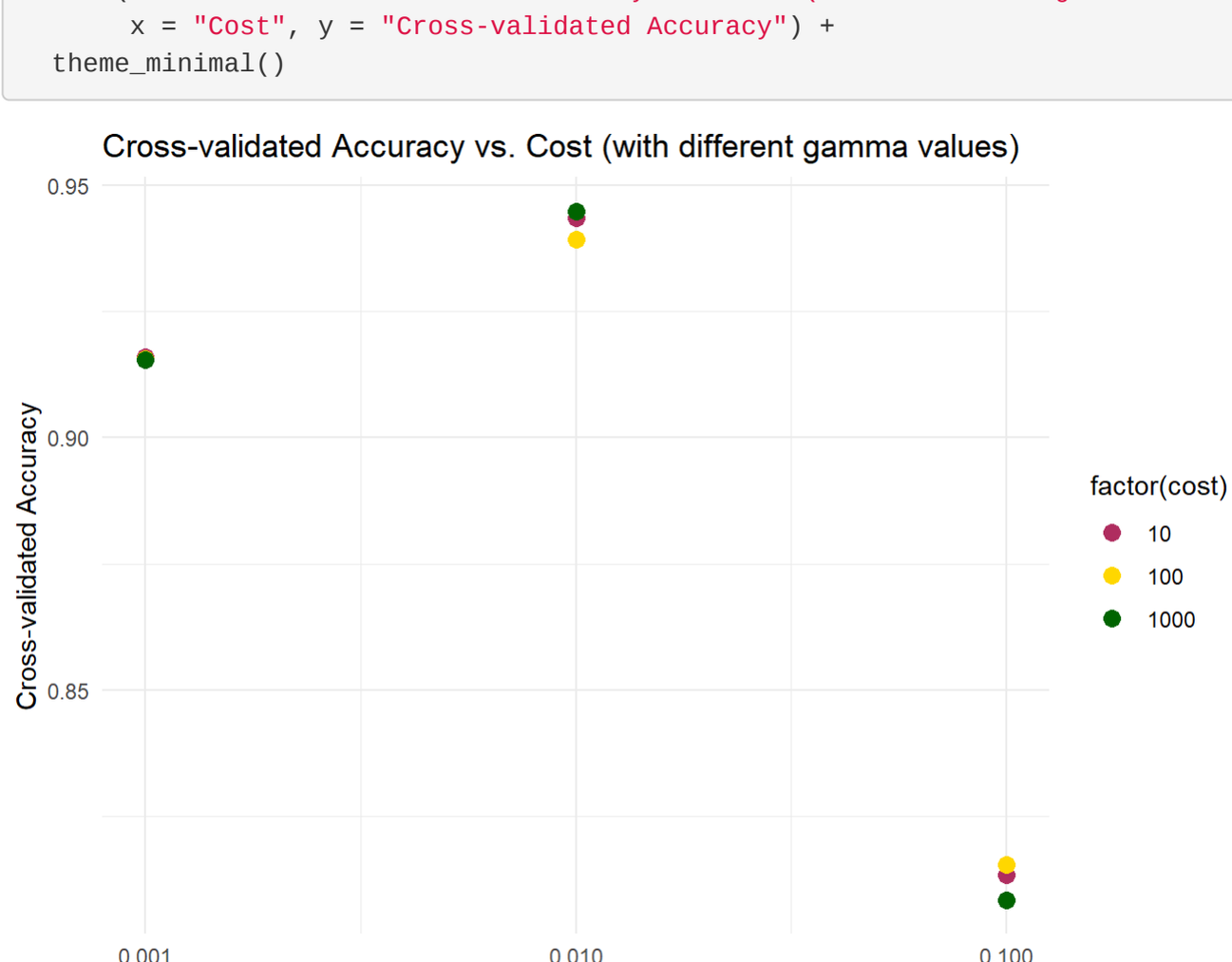
```
#For the time purposes I have decided to include only 10% of the data in our Cross-Validation wi
th all 10 classes. Moreover based on previous results I have assumed that cost must be greater o
r equal than 10 and that gamma has to be less or equal than 0.1. Hence I have decided to do 3 va
lues for cost (10, 100, 1000) and 3 for gamma (0.001, 0.01, 0.1).

Test_Prep <- dataPrep(0.1, 10, images.test, labels.test)

Test_Data <- createMatrix(Test_Prep)

Train_Prep <- dataPrep(0.1, 10, images.train, labels.train)

Train_Data <- createMatrix(Train_Prep)

library(e1071)
library(foreach)
library(doParallel)

# Combine predictor variables and response variable into a data frame
df <- data.frame(Train_Data[[1]], y = Train_Data[[2]])

#rm(list=setdiff(ls(), "df"))

# Define the tuning control (e.g., cross-validation)
tc <- tune.control(cross = 2)  # 5-fold cross-validation

# Define the range of values for the cost and gamma parameters
cost_range <- c(10, 100, 1000)
gamma_range <- c(0.001, 0.01, 0.1)

# Define the number of cores for parallel processing
num_cores <- detectCores() - 1  # Use all available cores except one for the OS

# Register parallel backend
registerDoParallel(cores = num_cores)

# Perform hyperparameter tuning in parallel
tune.out <- foreach(cost = cost_range, .combine = rbind) %:%
  foreach(gamma = gamma_range, .combine = rbind) %dopar% {
    # Perform hyperparameter tuning for the current combination of cost and gamma
    tune_results <- tune(svm, y ~ ., data = df,
                         kernel = "radial", ranges = list(cost = cost, gamma = gamma),
                         tunecontrol = tc)

    cv_accuracy <- 1 - tune_results$best.performance

    # Return the cost, gamma, and cross-validated accuracy
    c(cost = cost, gamma = gamma, cv_accuracy = cv_accuracy)
  }

# Stop parallel backend
stopImplicitCluster()

# Print the results
print(tune.out)
```

```
##          cost gamma cv_accuracy
## result.1   10 0.001   0.9160000
## result.2   10 0.010   0.9435000
## result.3   10 0.100   0.8133333
## result.1  100 0.001   0.9155000
## result.2  100 0.010   0.9391667
## result.3  100 0.100   0.8155000
## result.1 1000 0.001   0.9153333
## result.2 1000 0.010   0.9448333
## result.3 1000 0.100   0.8085000
```

```
#Define colors
custom_colors <- c("maroon", "#FFD700", "darkgreen")

# Create the plot
ggplot(tune.out, aes(x = cost, y = cv_accuracy, color = factor(gamma), group = factor(gamma))) +
  geom_point(size = 3) +
  scale_color_manual(values = custom_colors) +
  scale_x_log10() +  # Use log scale for the x-axis (cost)
  labs(title = "Cross-validated Accuracy vs. Cost (with different gamma values)",
       x = "Cost", y = "Cross-validated Accuracy") +
  theme_minimal()
```




```
ggplot(tune.out, aes(x = gamma, y = cv_accuracy, color = factor(cost), group = factor(cost))) +
  geom_point(size = 3) +
  scale_x_log10() +  # Use log scale for the x-axis (cost)
  scale_color_manual(values = custom_colors)+
  labs(title = "Cross-validated Accuracy vs. Cost (with different gamma values)",
       x = "Cost", y = "Cross-validated Accuracy") +
  theme_minimal()
```



As we can see, the best result is obtained for cost = 1000 and gamma = 0.01. Moreover we can observe, that cost plays little role compared to gamma. Let us now apply that knowledge to a model trained on half of the original data set and compare it with results from subpoint 10.

```
Test_Prep <- dataPrep(0.5, 10, images.test, labels.test)

Test_Data <- createMatrix(Test_Prep)

Train_Prep <- dataPrep(0.5, 10, images.train, labels.train)

Train_Data <- createMatrix(Train_Prep)

new_model_results <- fitAndTest(Train_Data, Test_Data, cost_in = 1000, gamma_in = 0.01) #Optimal
                                                                                         #paramet
rization
old_model_results <- fitAndTest(Train_Data, Test_Data, cost_in = 1, gamma_in = 0.05) #Oryginal p
arametrization

c(new_model_results, old_model_results)
```

```
## [1] 0.9794 0.9772
```

As we can see, the results for the model indicated by cross validation is doing slightly better than the old model. We may explain it as a very small difference in gamma, which seems to be the leading parameter here (as shown on the figures).